k-Selection

Yufei Tao

Department of Computer Science and Engineering
Chinese University of Hong Kong
In this lecture, we will put randomization to some real use, by using it to solve a non-trivial problem called k-selection elegantly and efficiently.
The *k*-Selection Problem

**Problem:** You are given a set $S$ of $n$ integers in an array, and also an integer $k \in [1, n]$. Design an algorithm to find the $k$-th smallest integer of $S$.

For example, suppose that $S = (53, 92, 85, 23, 35, 12, 68, 74)$, and $k = 3$. You should output 35.

This problem can be easily settled in $O(n \log n)$ time by sorting. Next, we will solve it in $O(n)$ expected time with randomization.
To illustrate the idea behind our algorithm, suppose that we pick an arbitrary element (say the first) \( v \) of \( S \).

Move elements around so that those smaller than \( v \) are placed before \( v \), and those larger are placed after \( v \). This requires only \( O(n) \) time (no sorting required).

- If \( x = k - 1 \), done—\( v \) is what we are looking for!
- If \( x < k - 1 \), recurse by performing \((k - (x + 1))\)-selection on the \( y \) elements to the right of \( v \).
- If \( x > k - 1 \), recurse by performing \( k \)-selection on the \( x \) elements to the left of \( v \).
Idea

Obstacle: $x$ or $y$ can be very small (0 if we are unlucky) such that we can throw away only few elements before recursion!

\[
\begin{array}{ccc}
< v & v & > v \\
\end{array}
\]

$x$ elements $y$ elements

Wish: Make $x \geq n/3$ and $y \geq n/3$.
Anecdote: Randomly select $v$ from the whole array! Wish comes true with probability 1/3!

New obstacle: Would still fail with probability 2/3.
New anecdote: Choose another $v$ if we fail—3 repeats in expectation!
The **rank** of an integer $v$ in $S$ is the number of elements in $S$ smaller than or equal to $v$.

For example, suppose that $S = (53, 92, 85, 23, 35, 12, 68, 74)$. Then, the rank of 53 is 4, and that of 12 is 1.

Finding the rank of $v$ in $S$ (stored in an array) takes only $O(|S|)$ time.
Algorithm

1. Randomly pick an integer $v$ from $S$.
2. Get the rank of $v$—let it be $r$.
3. If $r$ is not in $[n/3, 2n/3]$, repeat from Step 1.
4. Otherwise:
   4.1 If $k = r$, return $v$.
   4.2 If $k < r$, produce an array $A$ containing all the integers of $S$ strictly smaller than $v$. Recurse on $A$ by looking for the $k$-th smallest element in $A$.
   4.3 If $k > r$, produce an array $A$ containing all the integers of $S$ strictly larger than $v$. Recurse on $A$ by looking for the $(k - r)$-th smallest element in $A$. 
Example

Consider that we want to find the 10th smallest element from a set $S$ of 12 elements:

\[
\begin{array}{c c c c c c c c c c c c}
17 & 26 & 38 & 28 & 41 & 72 & 83 & 88 & 5 & 9 & 12 & 35 \\
\end{array}
\]

Suppose that the $v$ we randomly choose is 12, whose rank is 3. This is not in the range of [4, 8]

So we repeat by randomly choosing another $v$ from $S$. Suppose that this time $v = 83$, whose rank is 11. This is not good either.

Repeat by choosing yet another $v$, say, 35, whose rank is 7. We generate an array with only the elements larger than 35:

\[
\begin{array}{c c c c c c c c c c c c}
38 & 41 & 72 & 83 & 88 \\
\end{array}
\]

Recurse by finding the 3rd smallest element in this array.
Cost Analysis

Step 1 (on Slide 7) takes $O(1)$ time.
Step 2 takes $O(n)$ time.

How many times do we have to repeat the above two steps?
With a probability $1/3$, we can proceed to Step 3 $\Rightarrow$ need to repeat only 3 times in expectation!

When we are at Step 3, $A$ has at most $\lceil 2n/3 \rceil$ elements left.
Cost Analysis

Let $f(n)$ be the expected running time of our algorithm on an array of size $n$.

We know from the earlier analysis:

\[
\begin{align*}
    f(1) &\leq O(1) \\
    f(n) &\leq O(n) + f(\lceil 2n/3 \rceil).
\end{align*}
\]

Solving the recurrence gives $f(n) = O(n)$ (master theorem).
It is worth mentioning that the k-selection problem can actually be solved in $O(n)$ time deterministically. However, the algorithm is much more complicated—this demonstrates again the power of randomization.